TECHNOLOGY GUIDE

Intel Corporation

AF_XDP Sockets: High Performance Networking for Cloud-Native Networking Technology Guide

Author

Magnus Karlsson

1 Introduction

Cloud-native networking is rapidly taking off and, in some areas, replacing network function virtualization as the preferred choice to deploy and scale networking workloads in the cloud. Developers have to make a choice on what interface to use to access the network.

Developers who use cloud-native network function will find an overview of AF_XDP and when it is viable as an interface for networking traffic.

Cloud-native technologies and principles are a good way to scale network workloads called cloud-native network functions (CNFs) to large cloud scales. This technology is rapidly replacing network function virtualization (NFV) and its virtual network functions (VNFs) based on virtualization technologies. NFVs are hard to scale, slow to upgrade, and slow to restart. Cloud native, on the other hand, uses Linux containers, which are simply regular processes with a limited view of the operating system and its resources. One major difference between a CNF and a VNF is that with a CNF you generally have no control over the operating system it runs on. Your CNFs can coexist with other CNFs and processes on the same system or even on the same core in a public cloud provider. For cloud native to provide the required security and coexistence requirements, it has to preserve the Linux security model, not require a specific execution model, and not require resources that might not exist or already be claimed by something else. Basically, it always works independently on the environment. The problem is that current technologies for providing high-speed raw packet networking are based on SR-IOV and SIOV combined with user-space drivers, and these require specific resources and execution models that are generally not possible to guarantee in a public cloud system. The question is, then, how can we provide high-speed networking for a CNF that is not based on SR-IOV or SIOV coupled with user-space drivers?

This technology guide shows how Address-Family eXpress Data Path (AF_XDP), a new socket type debuted in Linux 4.18, can be used to provide high-performance raw packet networking into a CNF without requiring any dedicated resources, specific execution models, or relaxing of the Linux security model. AF_XDP takes some of the key learnings from the Data Plane Development Kit (DPDK) on how to provide fast packet delivery and applies them to Linux, while excluding the techniques that would sacrifice security, or require dedicated resources or specific execution models. As shown in the Performance section, AF_XDP, in Linux 5.8 that is shipped with Ubuntu 20.10, can reach more than 10 Mpps for 64-byte packets. This is one order of magnitude above previous raw socket interfaces, but less than what a user-space driver would be able to achieve. AF_XDP does require some additions to each Linux networking driver to perform at its highest speed. Fortunately, it is supported by most Intel network interface cards (NICs), some NICs from other vendors, as well as support that is being developed for cloud providers' environments.

This document is part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits.

Table of Contents

Figures

Tables

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
1C1T	One thread on one core
1C2T	Two hyper-threads on the same core
2C2T	Two threads on two different cores
AF_PACKET	AF_PACKET was the main raw socket interface in Linux before AF_XDP was introduced.
AF_XDP	Address Family eXpress Data Path, a socket type introduced in Linux 4.18
CNF	Cloud-Native Network Function
DPDK	Data Plane Development Kit (DPDK)
eBPF	extended Berkeley Packet Filter
IA	Intel® architecture
IOMMU	Input/Output Memory Management Unit
NFV	Network Function Virtualization
NIC	Network Interface Card
OVS	Open vSwitch
PMD	Pole Mode Driver
SIOV	Scalable I/O Virtualization
SR-IOV	Single-Root Input/Output Virtualization
TSO	TCP Segmentation Offload
VNF	Virtual Network Function
VPP	Vector Packet Processing
XDP	eXpress Data Path

2 Overview

AF_XDP was introduced in Linux 4.18 and is a new socket type for fast, raw packet data delivery to user-space processes. It is present in most Linux distributions including Red Hat and Ubuntu. One key design choice of AF_XDP is that it uses Linux kernel drivers and not any user-space drivers. To achieve high performance and good scalability, it is based on lockless rings, batching, low use of interrupts, zero-copy semantics, and optimized networking drivers in the same vein as high-performance user-space packet processing libraries such as DPDK and VPP. While it is possible to use AF_XDP on any driver, it only achieves high performance levels if the Linux driver has zero-copy support added and has been optimized for this.

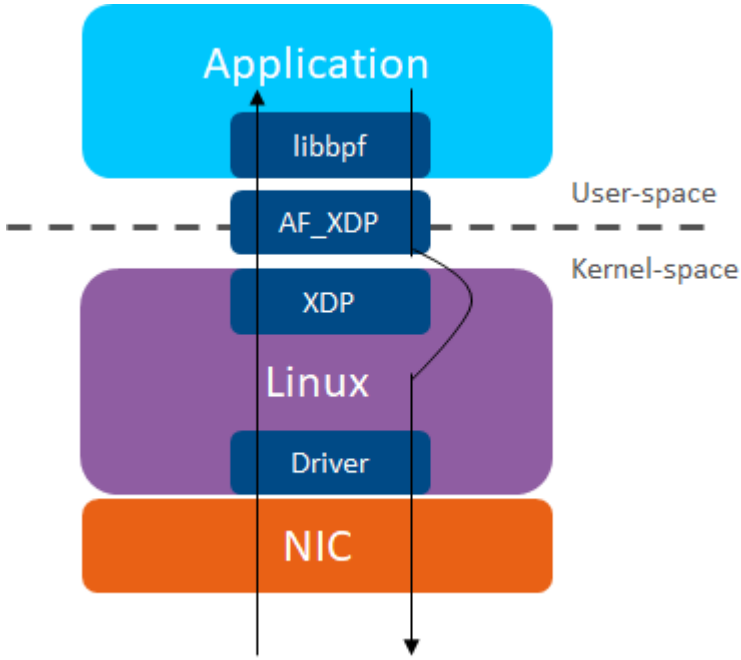

Figure 1. Flow of a Packet of AF_XDP

The flow of a packet of AF_XDP is shown in Figure 1. The packet first arrives at the NIC and, after some time, a NIC-specific driver in Linux accesses the NIC hardware to get the packet. After that, the packet passes through eXpress Data Path (XDP). This consists of an extended Berkeley Packet Filter (eBPF) program that is executed inside the driver and that is loaded from the user-space. This XDP program is executed once per packet and delivers a verdict for each packet: Send to the Linux stack, drop it, send it out some network interface card (NIC) interface, or send it up to user-space. Note that XDP is not used for egress traffic as it is currently only executed on ingress. The packet us sent to the user-space through an AF_XDP socket. Opening an AF_XDP socket makes it possible to receive raw packets straight from the NIC. In user-space there is also a library available in many distributions that is called libbpf. It contains helper functions that the application writer can use to facilitate development. The use of libbpf is strongly encouraged.

Initializing an AF_XDP socket using libbpf is as easy as calling two functions: one to register a packet buffer area where packets will be located and one to create and bind the socket to a networking interface. Packets from this interface will then be received in the registered packet area and packets sent from this packet area will be transmitted by that same networking interface. While AF_XDP can be used as is, it does not provide a packet buffer allocator that tracks which buffers are in flight and which ones are free. Existing packet processing libraries such as Data Plane Development Kit (DPDK) and Vector Packet Processing (VPP) provide this, and they both support AF_XDP. Several languages such as Rust and Google Go also have native support for AF_XDP and access interfaces.

3 AF_XDP Benefits in Cloud-Native Networking

What are the requirements for cloud-native networking and how can AF_XDP satisfy them?

- **Execution Environment Agnostic**
  The CNF should be independent of how the system is configured or what other processes are executed on it. In a private cloud, this requirement can be relaxed since the owner can tightly control its own system if it chooses to do so. But in a public cloud, this is generally not possible. AF_XDP has been written to not assume anything about the setup of the Linux system, nor where other processes are located or what they are doing in the system. The requirement is that there is at least one networking card and that AF_XDP has been enabled in the Linux kernel that is running. These choices have performance implications, and the trade-offs can be made by the user. AF_XDP can run in a system with and without hugepages, dedicated isolated cores, or an input/output memory management unit (IOMMU). You do not have to use threads and shared memory as it works with all Linux execution models including private memory models and processes. It is also independent of other processes that are running. You can, for example, safely run multiple packet processing processes on the same core with arbitrary scheduling modes. Basically, it is independent of how you use the rest of the system, and this is key in a public cloud as you do not have control of the system configuration or what other processes run there, unless you bought a whole dedicated node.
- **Binary Executable on Any Intel® Architecture (IA) Hardware**
  In the public cloud, your binary can be executed on any platform with the same instruction architecture. So, it is possible for your application to be moved to a system where the CPU, Linux kernel, or the NIC is newer than what you compiled for. It is therefore imperative that everything is binary compatible. Your binary should be able to be deployed on any hardware and system, including ones that did not exist when you created it. Linux guarantees binary compatibility and, as AF_XDP is part of

Linux, it does too. New IA features are dynamically chosen at runtime and not compiled into the binary so that binary compatibility is preserved.

- **Security**
  Security is paramount, especially in a potentially shared system. Linux is continuously improving its security with new features and patches. All these features work in conjunction with AF_XDP as it is an integral part of Linux. As stated previously, AF_XDP works with the process model that underlines most cloud-native orchestration systems and provides more security and protection, but generally lower performance, than threads based on shared memory.
- **Quick to Start and Stop**
  To be able to dynamically scale the workload as demand varies and to respond to failures, it is important to be able to start, stop, and spawn cloud-native network functions (CNFs) quickly. AF_XDP socket creation is quick and all structures are automatically cleaned up and reclaimed when the process exits. So, there is no need for any explicit cleanup.
- **Run Multiple Versions at the Same Time**
  In a large deployment, multiple versions of the CNF will exist at the same time, so there should be no restriction on how many versions of the same CNF can be run at the same time. There should be no need to upgrade all of them at the same time as this is completely infeasible in a large system. It would require the service to be shut down, which defeats the goal of being highly available. As AF_XDP is fully binary compatible and does not contain any runtime environment, this is not a problem. As many versions as desired can be executed at the same time.
- **Power Efficient**
  The number of packets that a CNF receives usually varies according to a pattern. It could be diurnal, based on events happening, or vary much more rapidly than that. It is therefore important to be able to save power or yield the core to some other CNF when the application has nothing or little to do. The usual Linux system calls like select(), poll(), and epoll() work with AF_XDP and can be used to sleep (and also yield to other processes, if needed) until a packet is received, can be sent, or a certain time has expired. Linux power-save governors work as usual and can be used to adjust the frequency and use the various power save states that IA offers. Performance monitoring tools will also show the actual percentage of time spent processing packets, Note that if maximum performance is desired and you are not concerned with power efficiency, then it is possible to busy-poll (i.e., constantly check in a loop if a packet has been received) with AF_XDP by using the functions that are provided in the user-space instead of the system calls above.
- **Deployable at Scale**
  As cloud systems tend to be large, or there is at least a hope that they will be scalable to large scales, it is important that the CNF is deployable at scale. There are many things that need to be in place to be able to do that, but here we are only going to mention one: simplicity. To deploy something at scale, it needs to be very simple and capable of working all the time on any system.
- **Performance**
  At the end, the system must have “good enough” performance. If the system crawls, none of the other properties matter as you cannot deploy your system anyway. Unfortunately, there is no such thing as eating the cake and keeping it at the same time. The cost of getting all of the properties above is that you get lower performance than if you used a user-mode driver with, for example, SR-IOV. If you are building an appliance, you generally do not care about most of the above properties as you have full control over your appliance box. In this case, it is likely a much better choice to use SR-IOV and user-mode drivers as they will provide you with a performance boost. As always, there is no reason to pay for things you do not need. The performance of AF_XDP is evaluated in the next section.

4 Performance

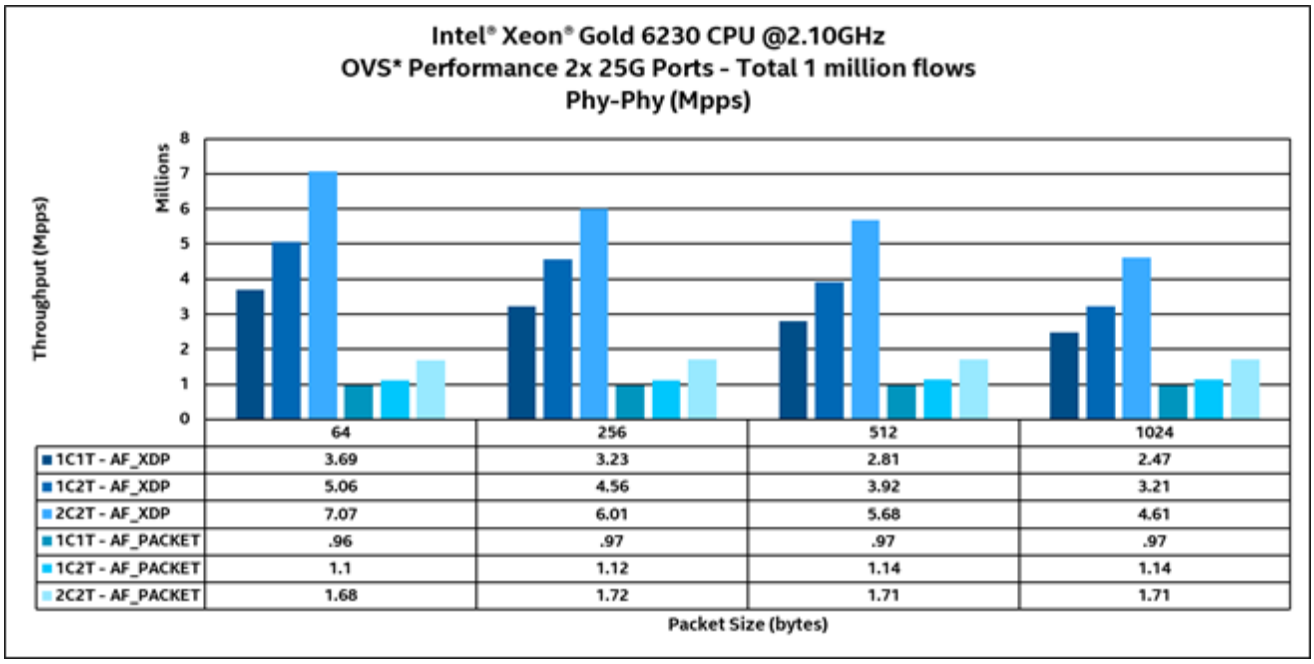

Figure 2. Throughput in Mpps of OVS for AF_XDP and AF_PACKET[1]

In Figure 2, the higher value is better. 1C1T means one thread on one core is used. 1C2T is the case where two hyper-threads on the same core are used, while 2C2T uses two threads on two different cores.

To give an example of the performance of AF_XDP in a hypothetical cloud system where you have no control over the operating system or the hardware resources, we ran Open vSwitch (OVS) on a standard Ubuntu 20.10 setup. The kernel is the vanilla Linux 5.8 kernel that comes with that version of Ubuntu. We did not modify the boot command line to use any hugepages, nor any isolated or dedicated CPUs. Important security features such as the IOMMU are left turned on. In other words, a standard system that you would likely see in a shared public cloud or the system you get when you boot up Ubuntu without any tunings or modifications. Note that this is not the system that will yield the highest throughput, as hugepages, isolated CPUs, and optimized kernel compile-time configurations will really help. But it is a good example of a system you will find that is not controlled or tuned by the application provider. As an upper bar for AF_XDP performance on this system setup, the simple l2fwd microbenchmark shipped with the kernel achieves around 11.5 Mpps for a 0% packet loss measurement.

Figure 2 shows the throughput at 0% packet loss for running OVS in the system described above. For detailed setup instructions, refer to the last paragraph in this section. AF_XDP delivers between 3.69 Mpps and 7.07 Mpps for 64-byte packets. This is three to five times more than AF_PACKET produces. AF_PACKET was the main raw socket interface in Linux before AF_XDP was introduced. You get a substantial performance increase by just changing the socket type.

Figure 3 shows the throughput at 0% packet loss for running OVS-DPDK with an AF_XDP DPDK poll mode driver (PMD) in the same system. By using the AF_XDP PMD of DPDK, any DPDK application can be run on AF_XDP without modification; in this case, it is OVS-DPDK. Here the throughput is between 4.34 Mpps and 8.66 Mpps for 64-byte packets. This is around 20% more than for the stock OVS.

Testing was performed on October 28, 2020. The system used was an Intel® Xeon® Gold 6230 processor 2.10 GHz with Intel® Turbo Boost Technology disabled. While it offers 80 CPUs with hyper-threading turned on, these experiments used a maximum of four, all located on the same socket as the NIC. The NIC was an Intel® Ethernet XXV710 DA2 Adapter with a total of two ports. OVS-DPDK was version 2.14.90 with a one-line patch to enable it to work with 4K pages and not assume that hugepages were always available. All measurements were performed according to RFC 2544 at 0% packet loss. See Appendix 1 for configuration details. For workloads and configurations visit www.Intel.com/PerformanceIndex. Results may vary.

[1] See backup for workloads and configurations. Results may vary.

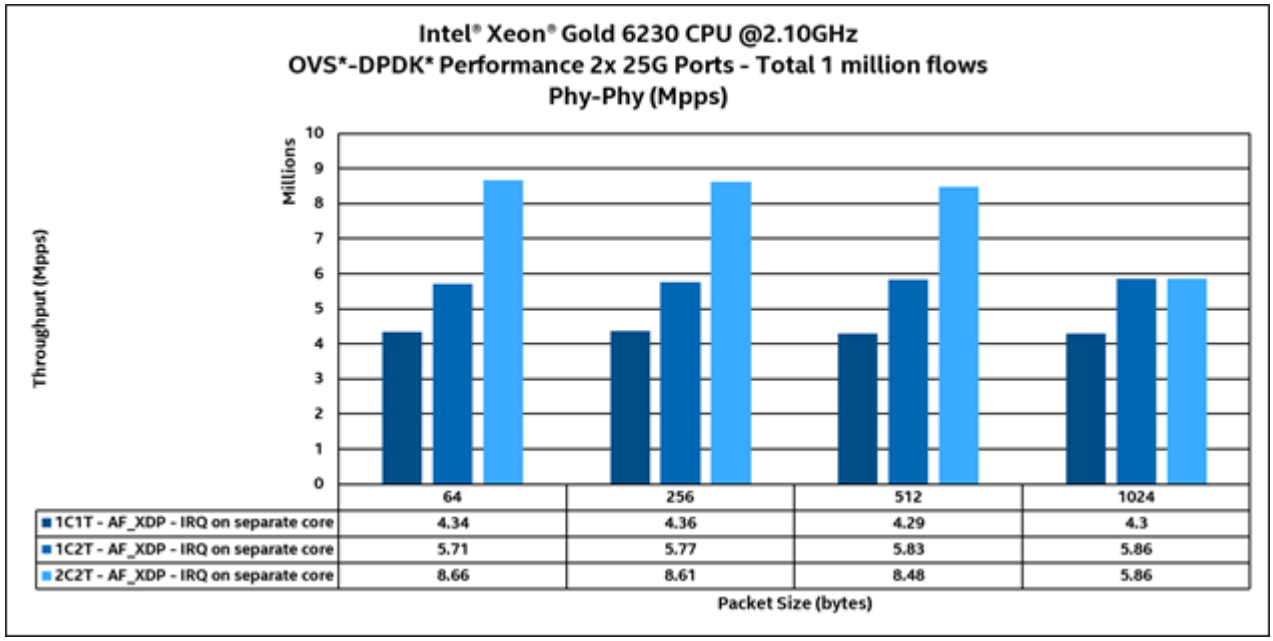

Figure 3. Throughput in Mpps of OVS-DPDK using the AFX_DP DPDK PMD[2]

Figure 3 shows the throughput in Mpps of OVS-DPDK using the AF_XDP DPDK PMD. The higher values are better. 1C1T means one thread on one core is used. 1C2T is the case where two hyper-threads on the same core are used, while 2C2T uses two threads on two different cores.

5 Ongoing and Future Work

AF_XDP is included with Linux 4.18, which was released in August 2018. Since then, a number of features have been added, such as DPDK support with the AF_XDP PMD, user-space helpers to facilitate using AF_XDP, statistics and debug support, and continuous performance improvements. One upcoming feature that there is a patch set for is support for jumbo frames that we hope will make it into the kernel. One missing feature that we would really like to get into the kernel is support for NIC metadata and offloading. Today, for example, you cannot read the timestamp of the packet through AF_XDP, nor can you use offloads like TSO.

6 Summary

Cloud-native networking with cloud-native network functions (CNFs) is increasingly popular as a scalable and secure way to deploy networking functions in a public or private cloud. This paper shows that AF_XDP, which has been is part of standard Linux since 4.18, provides a lot of benefits in these areas. AF_XDP is completely agnostic to the execution environment, is binary compatible, adheres to the Linux security model, is compatible with power-saving frameworks, and, in many cases, provides good enough performance to be a viable choice for providing high-speed network packet access for CNFs.

[2] See backup for workloads and configurations. Results may vary.

Appendix A Configuration Supporting Performance

The table in this section lists the configuration of the system used for testing. Testing was performed on October 28, 2020.

Table 2. Configuration Used During Performance Tests

	ITEM	DESCRIPTION
	Platform	Supermicro X11DPG-QT
CPU	Product	Intel® Xeon® Gold 6230
	Speed (MHz)	2100
	No of Cores	20
	Stepping	B0
	Level 1 Data Cache	32K
	Level 2 Data Cache	1024K
	LLC Cache	36424K
Memory	Vendor	Micron
	Type	DDR4
	Size	16384
	Channels	12
	Speed	2933
BIOS	Vendor	American Megatrends Inc.
	Version	3.3
	Date	02/21/2020
	Microcode	0x4002f01
Compiler Version	GCC Version	7_5_0
	Linker Version	2_30
	Assembly Version	2_30
Operating System	OS Version	Ubuntu_20_10
	Kernel Version	5.8.0-25
Benchmark Software	nic_firm	7.20
	nic_driver	The standard one in Linux 5.8.0-25
	DPDK Version	20.08.0
	OVS DPDK Version	2.14.90
	OVS Version	2.14.90

0121/DN/WIPRO/PDF 632920-001US